# Embedded Device Cooperative System Using Java Bytecode Instrumentation

Ryota AYAKI* , Kohei KADOWAKI* , Hideki SHIMADA** and Kenya SATO*

(Received January 19, 2010)

Recently, various embedded devices have become equipped with network functions for communication among themselves. In a network middleware with embedded devices (e.g. Jini) to cooperate with each other on networks, a device can use functions provided by other devices. However, because the function is defined by its interface, all programming interfaces of certain functions must be built into a client device beforehand so that the client device can use the functions. Therefore, already existing client devices cannot exploit newly released functions. In this paper, we propose an embedded device cooperative system to solve the problems by adapting dynamic program generation using Java bytecode instrumentation technology. We implemented the proposed system and evaluated its performance from processing time and memory usage perspectives.

**Key words** : network middleware, Java bytecode instrumentation, embedded device

## 1. Introduction

Recently, such various embedded devices as home appliances, mobile phones, and PDAs are often equipped with network functions for communication among themselves by using a network middleware (e.g. UPnP/DLNA [1], Jini [2], HAVi [3]). In the future these devices are expected to automatically and dynamically compose a network. New functions can be generated if embedded devices are connected to each other through a network. The network middleware is required to allow embedded devices to cooperate with other devices through a network regardless of hardware and OS. However, because a "function" is defined by its "interface", all programming interfaces of certain functions must be built into a client device beforehand so that the client device can use the functions. Therefore, already existing client devices cannot exploit newly released functions. Moreover, there is no method by which to apply a new function provided by another device to the device's input operation when using the function, because function expandability is not defined in the network middleware. In this paper, we propose an embedded device cooperative system to solve the problems with the current network middleware with embedded devices by adapting dynamic program generation using Java bytecode instrumentation technology.

## 2. Network middleware with embedded devices

### 2.1 *Network architecture*

The network middleware for cooperation among devices on networks (e.g. Jini), features low platform dependency because it is a Java-based technology. A function provided by a device with the Java-based network middleware is defined as a "service". A device can cooperate mutually with other devices using services. The network middleware is composed of the following four elements: Service Provider (device providing a service), Client (device using a service), Lookup Service (device managing all services on networks), and Codebase (HTTP server device containing files required for using services). Figure 1 shows the network architecture for cooperation among devices when a client uses a service provided by a service provider.

* Graduate School of Science and Engineering, Doshisha University, Kyoto
Telephone:+81-774-65-7564, Fax:+81-774-65-6801, E-mail:dtj0703@mail4.doshisha.ac.jp
** Faculty of Science and Engineering, Doshisha University, Kyoto

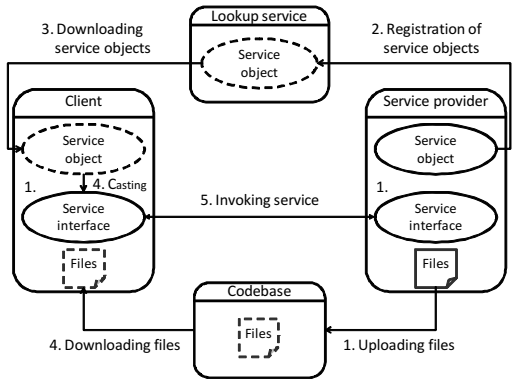


Fig. 1. Current network architecture for cooperation among devices.

1. Advance preparation

   A client needs to prepare a unique interface in Java language specification called "service interface" for using a service. Moreover, a service provider must store necessary files in a codebase so that the client can invoke the service.

2. Registration of service objects

   The service provider registers a service object generated from the service provided on a lookup service.

3. Downloading service objects

   The client downloads the desired service object from the lookup service.

4. Deserializing service objects

   The service object is serialized when it is transferred from the service provider to the client on networks. Therefore, the service object must be deserialized so that the client can use the services. The client can deserialize the service object with casting in Java language specification it using the service interface and the necessary files downloaded from the codebase.

5. Invoking service

   The client and the service provider communicate with each other by Remote Method Invocation (RMI), and the client invokes the service provided by the service provider with functions defined by the service interface.

### 2.2 *Problems*

The current network middleware with embedded devices has two problems with cooperation among devices and managing services on networks.

The first is that beforehand, all programming interfaces of certain functions must be built into a client device so that the client device can use the functions, because the function is defined by its interface. Therefore, already existing client devices cannot exploit newly released functions. In this paper, we call this the "adapting to undefined services problem".

The second is that there is no method by which to apply a new function provided by another device to an input operation of the device when using the function, because function expandability is not defined in the network middleware. Therefore users cannot apply the desired functions to desired input units. In this paper, we call this the "applying functions to input operation problem".

## 3. Proposed system

### 3.1 *Java bytecode instrumentation*

In this paper, we propose an embedded device cooperative system to solve the problems by adapting dynamic program generation using Java bytecode instrumentation technology. Generally, it is necessary to re-compile after modifying the source code (.java) for changing the bytecode (.class) in Java. However, Java bytecode instrumentation can directly modify the bytecode without re-compile. The proposed system can dynamically generate and modify the necessary programs for invoking the service by using it. We explain the proposed system using Figure 2, which shows the three devices: "device C with client function", "device S with service provider function", and "device L with lookup service".

In the proposed system, since all devices need to transfer files, they must have the codebase function in the network middleware. Device L must have Java bytecode instrumentation library. Moreover, device C needs to prepare "operation template" and device S needs to prepare "function block" as follows.

The methods invoked when device C's input unit is operated are defined in the operation template. For instance, when buttons are input units of

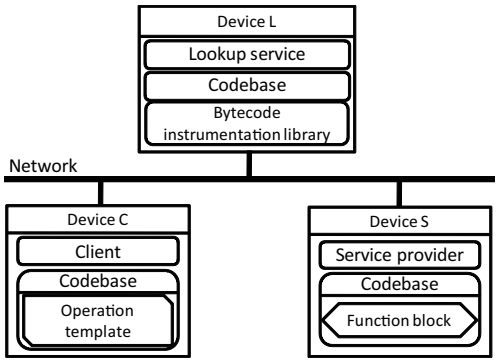


Fig. 2. Composition of Java bytecode instrumentation.

device C, the methods of the same number as the buttons in the operation template must be defined, and each button corresponds to one method as an event handler when operated. Information can be shown about the input unit of device C to other devices and users with the operation template.

The methods to invoke the functions provided by device S are defined in the function block. For instance, when there is a speaker service as a service provider, the former provides the function "play music". If the function is executed by the play method, the method must be defined in the function block, which can be used to show information about the function provided by device S to other devices and users.

These two programs only define the input operation of device C and the function provided by device S. Device L with the Java bytecode instrumentation library needs to build the functions defined by the function block into an operation template based on adaptation information chosen by users.

### 3.2 *Operation procedure*

In the current network middleware, cooperation between client and service provider device is possible by transferring a service object provided by the service provider to the client through the lookup service. On the other hand, the proposed system does not need to transfer service objects. Instead, in the proposed system, the client in device C and the service provider in device S must register URLs that indicate the operation template and the function block stored in their codebases to the lookup service in device L.

Figure 3 shows the operation procedure and the details when a client uses a service provided by a service provider.

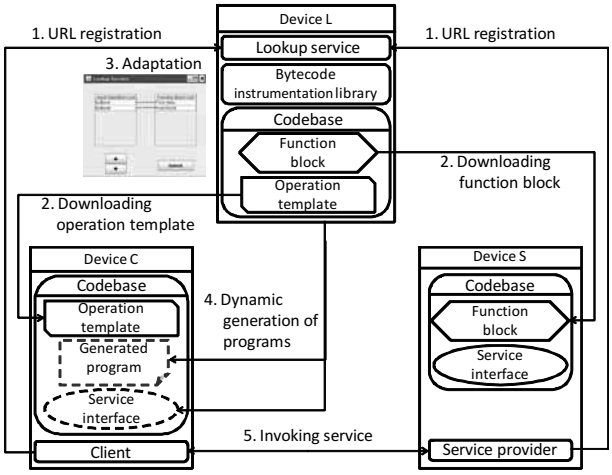


Fig. 3. Operation procedure of proposed system.

1. URL registration

   A client of device C registers a URL that indicates an operation template in a codebase in device L. Similarly, a service provider in device S registers a URL that indicates a function block.

2. Downloading files

   The lookup service in device L downloads the operation template and the function block from each codebase based on the registered URLs.

3. Applying functions to an operation template

   Device L displays the operation list of device C obtained from the operation template and the function list of device S obtained from the function block on the GUI. Users apply the desired functions to the input operation by GUI.

4. Dynamic generation of programs

   With the Java bytecode instrumentation library, device L builds the functions defined by the function blocks into the operation template based on adaptation information decided by users. Moreover, device L generates the necessary service interface for invoking the service based on the function blocks. Then the operation template that is dynamically built in functions and the service interfaces are stored in the codebase of device C.

5. Invoking service

   Device C communicates with device S with RMI using the generated programs. At this time, when users operate device C's input unit, device S's function defined in the service interface can be invoked.

## 4. Evaluation

### 4.1 *Implementation*

We implemented our proposed system by connecting three computers with the functions of devices C, S, and L with the system configuration explained in Section 3. Table 1 shows the evaluation environment of the proposed system. It includes BCEL [4], ASM [5], and Javassist [6] as a Java bytecode instrumentation library. We adopted Javassist because its abstraction level of API is high and it internally maintains the compiler.

Table 1. Evaluation environment.

| | |
|---|---|
| OS | Windows XP Professional SP2 |
| CPU | Pentium 4, 3.00 GHz |
| Memory | 1 GBytes |
| Java | JDK 1.6.0_05 |
| Jini | Jini Starter Kit 2.1 |

### 4.2 *Processing performance*

In the proposed system, processing that generates the operation template that is dynamically built in functions and the service interface by Java bytecode instrumentation is added to solve the original problems. Therefore, to consider the practicality of the proposed system, we evaluated the generation processing on the lookup service device. Figure 4 shows the relation of the number of built-in functions and the processing time. Next, Figure 5 shows the relation of the number of built-in functions and memory usage.

### 4.3 *Analysis*

Implementation was based on the design of the proposed system, and we confirmed that the system worked as proposed. Figure 4 shows that processing time is long with an increase in the number of functions. In general, the embedded devices do not provide numerous functions because they are developed for a special purpose. We verified that the proposed system was practical from a processing time perspective because, when the number of built-in functions is 1000, it can perform them in as quickly as 656 msec. Moreover, Figure 5 shows that memory usage increased with an increase in the number of functions. We do not assume that memory usage increases rapidly when the number of functions is increased. However, we must investigate the implementation method to minimize memory usage so that devices with limited memory capacity can apply the proposed system.

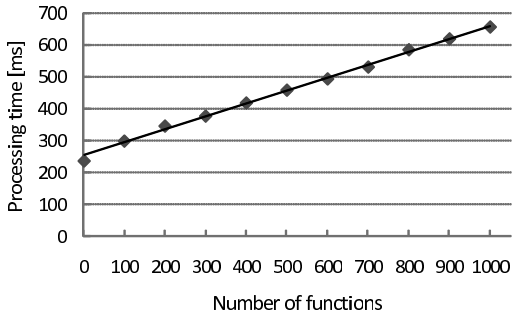


Fig. 4. Relation of the number of functions and the processing time.

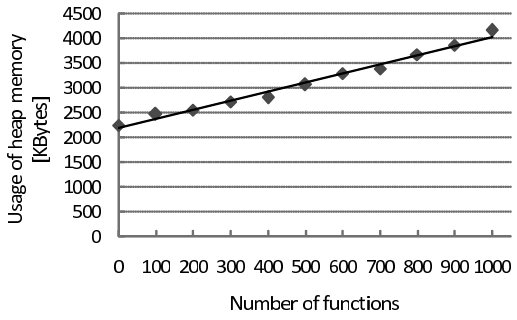


Fig. 5. Relation of the number of functions and the memory usage.

## 5. Related Work

In Adaptive Jini [7], which extends Jini's function, Jini's "adapting to undefined services problem" is solved by storing necessary programs for services in the service provider device's codebase. The client downloads it if necessary, and the "applying the func-

tion to input operation problem" is solved by providing and operating the client GUI. In systems using Adaptive Jini, the client device must have a display device because this system must do the input operation using the client's GUI.

## 6. Conclusion

In this paper, we proposed an embedded device cooperative system on networks and adopted Java-based network middleware to enable embedded devices to cooperate with each other. We successfully solved the following the original problems: "adapting to undefined services problem" and "show program lists" by adapting dynamic program generation using Java bytecode instrumentation technology on the original Jini. We also implemented the proposed system, evaluated its performance, and verified its practicability.

For future work, we will consider how to implement the proposed system on real embedded devices to evaluate performance in real environments.

## References

1) UPnP Forum, "Universal Plug and Play Device Architecture Version 1.0" (2008). http://www.upnp.org/

2) J. Waldo, and K. Arnold, The Jini Specifications - Second Edition, (Addison-Wesley, Boston, 2001).

3) The HAVi Organization, "HAVi Specification Version 1.1" (2001). http://www.havi.org/

4) The Apache Jakarta Project, "Byte Code Engineering Library" (2003). http://jakarta.apache.org/bcel/

5) E. Bruneton, R. Lenglet, and T. Coupaye, "ASM: a code manipulation tool to implement adaptable systems" (2002). http://asm.objectweb.org/current/asm-eng.pdf

6) S. Chiba, and M. Tatsubori, "Structural Reflection by Java Bytecode Instrumentation," IPSJ Journal, Vol.42, No.11, pp.2752-2760 (2001).

7) K. Kadowaki, H. Hayakawa, T. Koita, and K. Sato, "Design and Implementation of Adaptive Jini System to Support Undefined Services," Communication Networks and Services Research Conference, pp.577-583 (2008).